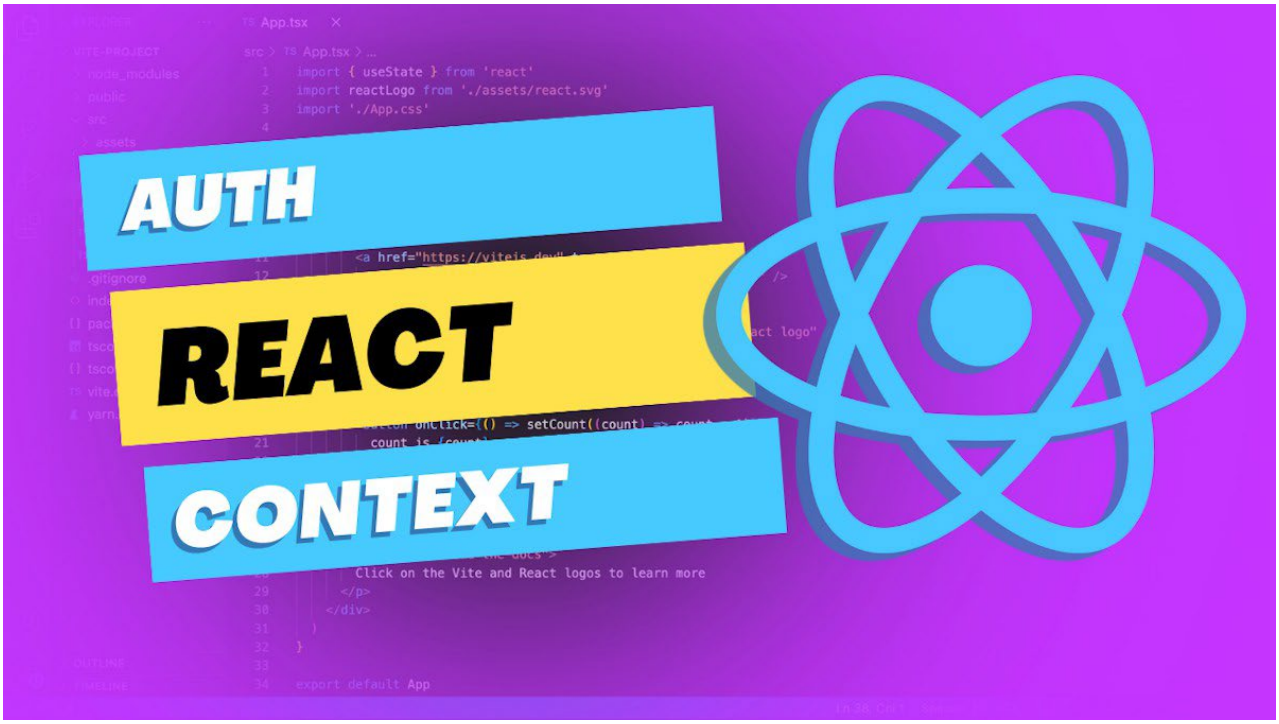
Auth & Context in React 18

sammeechward.com/use-context-auth



Imagina mostrar el nombre del usuario en el encabezado, mostrar sus publicaciones en la pantalla de inicio y revelar su foto de perfil en la barra lateral, todo mientras mantienes ciertas páginas fuera del alcance de los usuarios no autenticados.

En esta guía, aprenderá a compartir datos de usuario entre componentes con la API de contexto y el `enlace useContext`. También profundizaremos en la integración de esta poderosa técnica con **react-router** para proteger las rutas y actualizar la barra de navegación en función del estado de inicio de sesión del usuario.

Configuración del proyecto con Vite

Profundicemos en la creación de un nuevo proyecto de React 18 usando Vite. El objetivo aquí es crear una aplicación que pueda controlar la autenticación de usuarios mediante `useContext`. Así que, abróchense los cinturones de seguridad y ¡comencemos!

Cree un nuevo proyecto de Vite utilizando la plantilla React.

```
npx create-vite my-auth-app --template react
cd my-auth-app
npm install
```

Ahora que nuestro proyecto está listo, agreguemos los componentes necesarios y configuremos nuestro contexto de autenticación.

Creación del contexto de autenticación

Lo primero es lo primero, vamos a crear un contexto que administre el estado de inicio de sesión del usuario y almacene su token JWT. Lo llamaremos `AuthContext`.

Nota

El uso de Context nos permite realizar un seguimiento de la información con `useState` o `useReducer`, luego cualquier componente puede acceder a esa información sin tener que pasarla como props. Si no estás familiarizado con el funcionamiento de `userContext` , consulta la documentación oficial.

Cree un nuevo archivo src/AuthContext.jsx y agregue el siguiente contenido:

```
import { createContext, useState } from "react";

const AuthContext = createContext();

function AuthProvider(props) {
  const [isLoggedIn, setIsLoggedIn] = useState(false);
  const [token, setToken] = useState(null);

  const login = (jwtToken) => {
    setIsLoggedIn(true);
    setToken(jwtToken);
  };

  const logout = () => {
    setIsLoggedIn(false);
    setToken(null);
  };

  const value = {
    isLoggedIn,
    token,
    login,
    logout,
  };

  return <AuthContext.Provider value={value} {...props} />;
}

export { AuthContext, AuthProvider };
```

Esto es lo que está sucediendo en el código anterior:

1. Importa `createContext` y `useState` de React.
2. Cree un nuevo contexto denominado `AuthContext`.
3. Defina un componente de `función AuthProvider` que envolverá toda nuestra aplicación. Este componente mantiene el `isLoggedIn` y el estado `del token`.
4. Defina las funciones de `inicio` y `cierre de sesión` para administrar el estado y el token de inicio de sesión del usuario.
5. Pase el valor de contexto como un objeto que contiene las variables y funciones de estado.
6. Finalmente, exporte el `AuthContext` y el `AuthProvider` para que podamos usarlos en otras partes de nuestra aplicación.

Ahora podemos hacer que cualquier componente pueda acceder a la función `isLoggedIn,token,login` y `logout`.

Creación de la pantalla de inicio de sesión

Vamos a crear una pantalla de inicio de sesión muy simple que simula una solicitud de red para un token JWT.

Cree un nuevo archivo src/Login.jsx y agregue el siguiente contenido:

```
import { useContext, useState } from "react";
import { AuthContext } from "./AuthContext";

function Login() {
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const { login } = useContext(AuthContext);

  const handleSubmit = async (event) => {
    event.preventDefault();
    // Fake network request for JWT token
    const jwtToken = "fake-jwt-token";
    login(jwtToken);
  };

  return (
    <div>
      <h1>Login</h1>
      <form onSubmit={handleSubmit}>
        <label htmlFor="email">Email:</label>
        <input
          type="email"
          id="email"
          value={email}
          onChange={(e) => setEmail(e.target.value)}
        />
        <br />
        <label htmlFor="password">Password:</label>
        <input
          type="password"
          id="password"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
        />
        <br />
        <button type="submit">Login</button>
      </form>
    </div>
  );
}

export default Login;
```

En el componente `de inicio de sesión` anterior, nosotros:

1. Importe los ganchos necesarios y `AuthContext`.

2. Utilice el `gancho useContext` para acceder a la función de inicio de sesión desde nuestro `AuthContext`.
3. Defina el estado local para los campos de entrada de correo electrónico y contraseña.
4. Defina una `función handleSubmit` que simule una solicitud de red para un token JWT y llame a la función de `inicio de sesión` con el token falso.
5. Renderice un formulario simple con campos de entrada de correo electrónico y contraseña y un botón de envío.

Creación de la pantalla de inicio

Ahora vamos a crear una pantalla de inicio básica que muestre el estado de inicio de sesión del usuario y le permita cerrar la sesión.

Cree un nuevo archivo src/Home.jsx y agregue el siguiente contenido:

```
import { useContext } from "react";
import { AuthContext } from "./AuthContext";

function Home() {
  const { isLoggedIn, logout } = useContext(AuthContext);

  return (
    <div>
      <h1>Home</h1>
      {isLoggedIn ? (
        <>
          <p>Welcome! You are logged in.</p>
          <button onClick={logout}>Logout</button>
        </>
      ) : (
        <p>Please log in to access more features.</p>
      )}
    </div>
  );
}

export default Home;
```

En el componente `Inicio` anterior, nosotros:

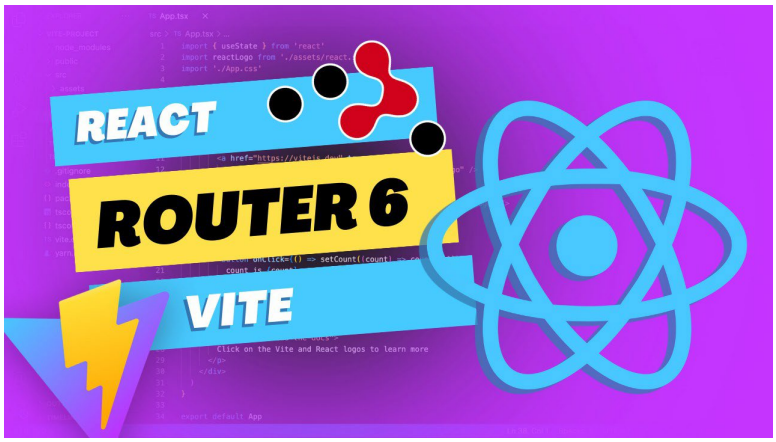
1. Importe los hooks necesarios y `AuthContext`.
2. Utilice el `enlace useContext` para acceder a `isLoggedIn` y `cerrar sesión` funciones de nuestro `AuthContext`.

1. Muestre un mensaje de bienvenida y un botón de cierre de sesión si el usuario ha iniciado sesión, de lo contrario, muestre un mensaje solicitando al usuario que inicie sesión.

Configuración de rutas con react-router-dom

Antes de sumergirnos en el envoltorio de nuestra aplicación con `AuthProvider`, primero instalemos `react- router-dom` y configuremos rutas para nuestros `componentes Home` e `Login`.

Si no estás familiarizado con `react-router-dom`, echa un vistazo:



Adición de React Router 6

En este tutorial, exploraremos el proceso de agregar React Router 6 a una aplicación React 18 usando...

hace 1 año

Instale React Router 6 como una dependencia.

```
npm install react-router-dom
```

Ahora, modifique src/App.jsx para incluir las nuevas rutas:

```
import { BrowserRouter, Link, Route, Routes } from "react-router-dom";
import Home from "./Home";
import Login from "./Login";

function App() {
  return (
    <BrowserRouter>
      <nav>
        <ul>
          <li>
            <Link to="/">Home</Link>
          </li>
          <li>
            <Link to="/login">Login</Link>
          </li>
        </ul>
      </nav>

      <main>
        <Routes>
          <Route path="/" element={<Home />} />
          <Route path="/login" element={<Login />} />
        </Routes>
      </main>
    </BrowserRouter>
  );
}

export default App;
```

Así que tenemos una pantalla de inicio de sesión y una pantalla de inicio, pero si intentas ejecutar la aplicación ahora mismo, no funcionará. Esto se debe a que aún no hemos empaquetado nuestra aplicación con `AuthProvider` . Un componente debe estar anidado dentro de `AuthProvider` para poder acceder a los valores de contexto.

Empaquetado de la aplicación con AuthProvider

Con nuestras rutas en su lugar, vamos a envolver toda nuestra aplicación con `AuthProvider`. Esto hará que los datos del usuario estén disponibles para cualquier componente anidado dentro de él.

Modifique src/App.jsx para incluir AuthProvider:

```
import { BrowserRouter, Link, Route, Routes } from "react-router-dom";
import { AuthProvider } from "./AuthContext";
import Home from "./Home";
import Login from "./Login";

function App() {
  return (
    <AuthProvider>
      <BrowserRouter>
        <nav>
          <ul>
            <li>
              <Link to="/">Home</Link>
            </li>
            <li>
              <Link to="/login">Login</Link>
            </li>
          </ul>
        </nav>

        <main>
          <Routes>
            <Route path="/" element={<Home />} />
            <Route path="/login" element={<Login />} />
          </Routes>
        </main>
      </BrowserRouter>
    </AuthProvider>
  );
}

export default App;
```

Probar la aplicación

Ahora que todo está configurado, es hora de probar nuestra aplicación.

Inicie el servidor de desarrollo.

```
npm run dev
```

Abra su navegador y navegue hasta `http://localhost:5173`. Debería ver la página de inicio que muestra el mensaje "Inicie sesión para acceder a más funciones". Haga clic en el enlace "Iniciar sesión" en el menú de navegación e ingrese cualquier correo electrónico y contraseña para iniciar sesión. Después de iniciar sesión, puede volver a la página de inicio, que ahora mostrará un mensaje de bienvenida y un botón de cierre de sesión.

¡Y eso es todo! Hemos creado con éxito una aplicación que utiliza el gancho `useContext de React 18` para administrar la autenticación de usuarios. Recuerde que este tutorial usa un token JWT falso para simplificar, pero en un escenario del mundo real, querrá reemplazarlo con una solicitud de red adecuada a su servidor de autenticación.

Redirigir a la página de inicio después de iniciar sesión

Una vez que el usuario inicia sesión, se le debe dirigir a una página diferente. En este caso, los redirigiremos a la página de inicio.

Modifique src/Login.jsx para redirigir al usuario a la página de inicio después de iniciar sesión:

```
import { useContext, useState } from "react";
import { AuthContext } from "./AuthContext";
import { Navigate } from "react-router-dom";

function Login() {
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const { login, isLoggedIn } = useContext(AuthContext);

  const handleSubmit = async (event) => {
    event.preventDefault();
    // Fake network request for JWT token
    const jwtToken = "fake-jwt-token";
    login(jwtToken);
  };

  if (isLoggedIn) {
    return <Navigate to="/" />;
  }

  //...
```

Aquí estamos comprobando si el usuario ha iniciado sesión y, si lo está, lo redirigimos a la página de inicio utilizando el `componente Navigate` de `react-router-dom`. Podemos utilizar una técnica similar para redirigir al usuario a la página de inicio de sesión si **no ha** iniciado sesión.

Adding a Protected Route

Now that we have our `AuthProvider` and routes set up, let's add a protected route to our app. This route will only be accessible to authenticated users. We'll create a `Profile` component that will be our protected route.

Creación del componente de perfil

Cree un nuevo archivo src/Profile.jsx y agregue el siguiente contenido:

```
import { useContext } from "react";
import { AuthContext } from "./AuthContext";

function Profile() {
  const { token } = useContext(AuthContext);

  return (
    <div>
      <h1>Profile</h1>
      <p>Your secret token is: {token}</p>
    </div>
  );
}

export default Profile;
```

En el componente `Perfil` anterior, nosotros:

1. Importe los ganchos necesarios y `AuthContext`.
2. Use el gancho `useContext` para acceder al `token` desde nuestro `AuthContext`.
3. Representar el token secreto del usuario como parte de la información de su perfil.

Creación de una guardia de ruta

Para proteger la `ruta de perfil`, crearemos un componente de orden superior llamado `RouteGuard` que comprobará si el usuario ha iniciado sesión antes de renderizar el componente protegido.

Cree un nuevo archivo src/RouteGuard.jsx y agregue el siguiente contenido:

```
import { useContext } from "react";
import { Navigate } from "react-router-dom";
import { AuthContext } from "./AuthContext";

function RouteGuard({ children }) {
  const { isLoggedIn } = useContext(AuthContext);

  return isLoggedIn ? children : <Navigate to="/login" />;
}

export default RouteGuard;
```

En el componente `RouteGuard` anterior, nosotros:

1. Importe los ganchos necesarios, `navegue` desde `react-router-dom` y `AuthContext`.
2. Utilice el `gancho useContext` para acceder al valor `isLoggedIn` desde nuestro archivo `AuthContext`.
3. Compruebe si el usuario ha iniciado sesión y, si es así, procese el componente protegido. De lo contrario, redirija al usuario a la `página /login`.

Adición de la ruta protegida

Por último, vamos a añadir el `componente Perfil` como ruta protegida en nuestra `App` componente.

Modifique src/App.jsx para incluir la ruta de perfil y envuélvala con RouteGuard:

```
import { BrowserRouter, Link, Route, Routes } from "react-router-dom";
import { AuthProvider } from "./AuthContext";
import Home from "./Home";
import Login from "./Login";
import Profile from "./Profile";
import RouteGuard from "./RouteGuard";

function App() {
  return (
    <AuthProvider>
      <BrowserRouter>
        <nav>
          <ul>
            <li>
              <Link to="/">Home</Link>
            </li>
            <li>
              <Link to="/login">Login</Link>
            </li>
            <li>
              <Link to="/profile">Profile</Link>
            </li>
          </ul>
        </nav>

        <main>
          <Routes>
            <Route path="/" element={<Home />} />
            <Route path="/login" element={<Login />} />
            <Route
              path="/profile"
              element={
                <RouteGuard>
                  <Profile />
                </RouteGuard>
              }
            />
          </Routes>
        </main>
      </BrowserRouter>
    </AuthProvider>
  );
}

export default App;
```

Ahora, la ruta `de perfil` está protegida por el componente `RouteGuard`. Los usuarios que no hayan iniciado sesión serán redirigidos a la `página /login` cuando intenten acceder a la página `/profile` ruta.

Con esta configuración, ha implementado correctamente una ruta protegida en su

Aplicación React usando `AuthProvider` y `react-router-dom`. No solo puede compartir datos entre páginas con facilidad, sino que también puede crear una experiencia de usuario más segura al proteger las rutas confidenciales.

Mejorar la experiencia del usuario con una barra de navegación dinámica

Para que nuestra aplicación sea aún más atractiva y fácil de usar, vamos a crear una barra de navegación dinámica que actualice su contenido en función del estado de inicio de sesión del usuario.

Modifique el archivo src/App.jsx para incluir un componente de navegación independiente:

```
import { useContext } from "react";
import { BrowserRouter, Link, Route, Routes } from "react-router-dom";
import { AuthProvider, AuthContext } from "./AuthContext";
import Home from "./Home";
import Login from "./Login";
import Profile from "./Profile";
import RouteGuard from "./RouteGuard";

function Navigation() {
  const { isLoggedIn } = useContext(AuthContext);

  return (
    <nav>
      <ul>
        <li>
          <Link to="/">Home</Link>
        </li>
        {isLoggedIn ? (
          <>
            <li>
              <Link to="/profile">Profile</Link>
            </li>
          </>
        ) : (
          <li>
            <Link to="/login">Login</Link>
          </li>
        )}
      </ul>
    </nav>
  );
}

function App() {
  return (
    <AuthProvider>
      <BrowserRouter>
        <Navigation />
        <main>
          <Routes>
            <Route path="/" element={<Home />} />
            <Route path="/login" element={<Login />} />
            <Route
              path="/profile"
              element={
                <RouteGuard>
                  <Profile />
                </RouteGuard>
              }
            />
          </Routes>
        </main>
      </BrowserRouter>
    </AuthProvider>
  );
}
```

```
export default App;
```

En el componente `de navegación` anterior, nosotros:

1. Importe los ganchos necesarios y `AuthContext`.
2. Utilice el `gancho useContext` para acceder al valor `isLoggedIn` desde nuestro archivo `AuthContext`.
3. Renderice condicionalmente el enlace "Perfil" o "Inicio de sesión" en función del estado de inicio de sesión del usuario.

Ahora, la barra de navegación mostrará el enlace "Perfil" solo cuando el usuario haya iniciado sesión. De lo contrario, mostrará el enlace "Iniciar sesión".

Almacenamiento y carga de tokens JWT desde el almacenamiento local

En esta sección, mejoraremos nuestra aplicación almacenando el token JWT en el almacenamiento local del navegador. De esta manera, el estado de inicio de sesión del usuario persistirá en todas las sesiones y no tendrá que volver a iniciar sesión cada vez que visite el sitio.

En primer lugar, actualicemos nuestro `AuthContext` para almacenar el token JWT en el almacenamiento local cuando el usuario inicie sesión y cargue el token desde el almacenamiento local cuando se inicialice el contexto.

Modifique src/AuthContext.jsx para incluir el control del almacenamiento local.

```
import { createContext, useContext, useState, useEffect } from "react";

const AuthContext = createContext();

function AuthProvider({ children }) {
  const [isLoggedIn, setIsLoggedIn] = useState(false);
  const [jwt, setJwt] = useState(null);
  const [isLoading, setIsLoading] = useState(true);

  useEffect(() => {
    const storedJwt = localStorage.getItem("jwt");
    if (storedJwt) {
      setIsLoggedIn(true);
      setJwt(storedJwt);
    }
    setIsLoading(false);
  }, []);

  function login(token) {
    setIsLoggedIn(true);
    setJwt(token);
    localStorage.setItem("jwt", token);
  }

  function logout() {
    setIsLoggedIn(false);
    setJwt(null);
    localStorage.removeItem("jwt");
  }

  const authValue = {
    isLoggedIn,
    isLoading,
    login,
    logout,
  };

  return <AuthContext.Provider value={authValue}>{children}
</AuthContext.Provider>;
}

export { AuthProvider, AuthProvider };
```

En el componente `AuthProvider` actualizado, hemos realizado los siguientes cambios:

1. Se ha añadido un `gancho useEffect` que se ejecuta cuando se monta el componente. Este enlace comprueba si hay un token JWT almacenado en el almacenamiento local y, si es así, establece el `estado isLoggedIn` en `true` y actualiza el estado `jwt` con el token almacenado.
2. Se modificó la función de `inicio` de sesión para almacenar el token JWT en el almacenamiento local cuando el usuario inicia sesión.

1. Se modificó la función de cierre de sesión para eliminar el token JWT del almacenamiento local cuando el usuario cierra la sesión.

Cuando el usuario inicie sesión, el token JWT se almacenará en el almacenamiento local y su estado de inicio de sesión persistirá en todas las sesiones del navegador. Cuando vuelvan a visitar el sitio, su estado de inicio de sesión se restaurará automáticamente utilizando el token almacenado.

Dado que `useEffect` se ejecuta después de montar el componente, no sabremos si el usuario ha iniciado sesión hasta que los componentes ya se hayan cargado. Por lo tanto, debemos incluir un `estado isLoading` para indicar que aún se está determinando el estado de inicio de sesión del usuario.

Modifique src/RouteGuard.jsx para usar isLoading.

```
import { useContext } from "react";
import { Navigate } from "react-router-dom";
import { AuthContext } from "./AuthContext";

function RouteGuard({ children }) {
  const { isLoggedIn, isLoading } = useContext(AuthContext);

  if (isLoading) {
    return <></>;
  }

  if (!isLoggedIn) {
    return <Navigate to="/login" />;
  }

  return children;
}

export default RouteGuard;
```

1. Si el localStorage aún no se ha comprobado, `isLoading` será true y devolveremos un fragmento vacío para "no hacer nada".
2. Si el usuario no ha iniciado sesión, lo redirigiremos a la página de inicio de sesión.
3. De lo contrario, el usuario ha iniciado sesión y renderizaremos los componentes secundarios.

Código final

https://github.com/Sam-Meech-Ward/react-router-auth-context

Resumen

En este tutorial, hemos creado una aplicación React 18 con Vite que puede manejar la autenticación de usuarios mediante el gancho useContext. Hemos creado un AuthContext para administrar el estado de inicio de sesión y el token JWT, un componente de inicio de sesión para controlar los inicios de sesión de los usuarios y un componente Inicio para mostrar el estado del usuario. También agregamos una ruta protegida y una página secreta para demostrar cómo restringir el acceso en función del estado de inicio de sesión del usuario.

Como puedes ver, el enlace useContext es una herramienta eficaz para administrar y compartir el estado en toda la aplicación. Si bien este tutorial se centró en la autenticación de usuarios, los conceptos también se pueden aplicar a muchos otros casos de uso.